

Crash Course on Character Encodings

Yusuke Shinyama

NYCNLP

Oct. 27, 2006



Introduction



Are they the same?

- Unicode
- UTF

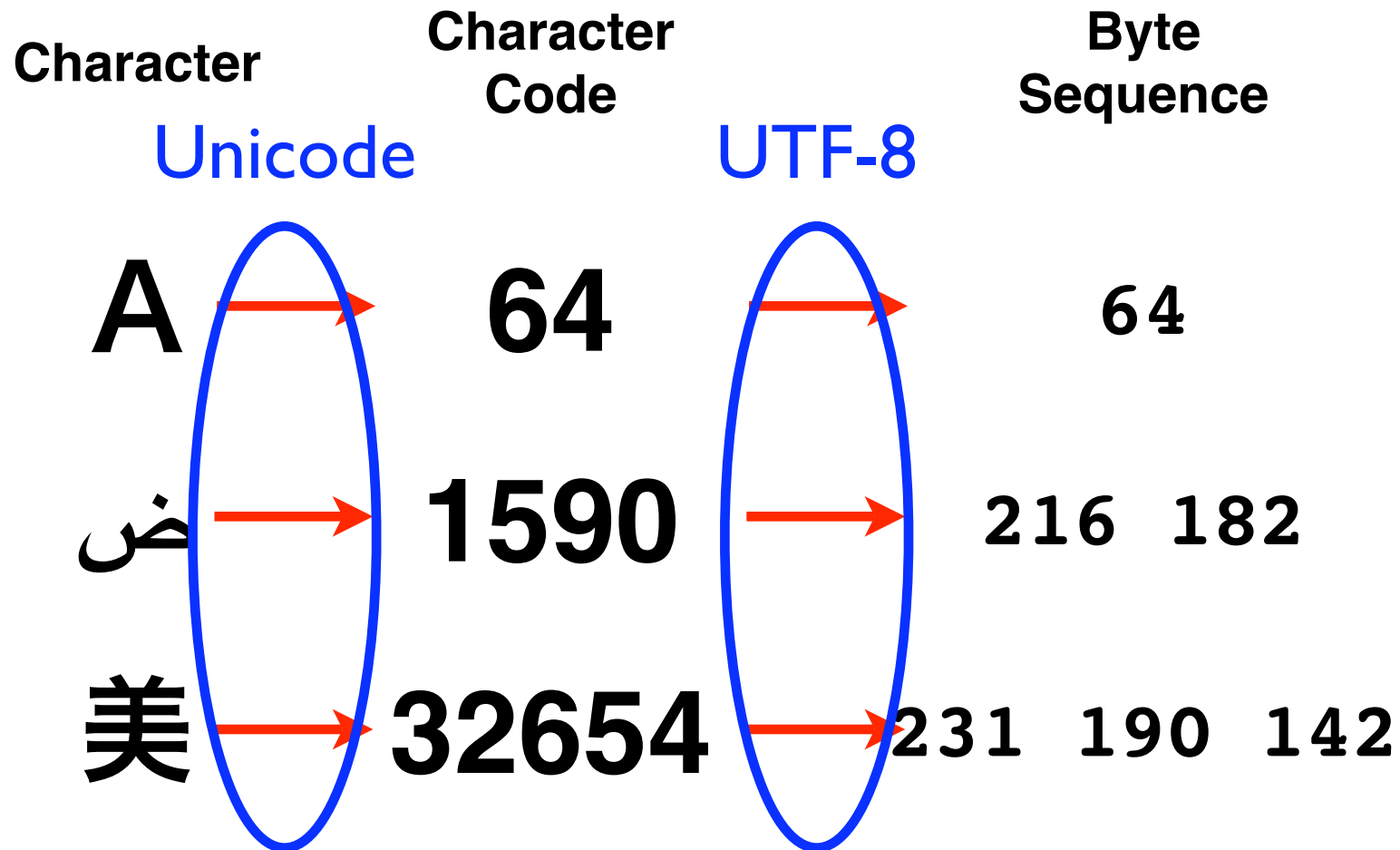


Two Mappings

Character	Character Code	Byte Sequence
A	64	64
ح	1590	216 182
美	32654	231 190 142



Two Mappings



“Character Set”

“Encoding Scheme”



Terminology

- **Character Set**

- Mapping from abstract characters to numbers.

- **Encoding Scheme**

- Way to represent (encode) a number in a byte sequence in a decodable way.
- Only necessary for character sets that have more than 256 characters.



In ASCII...

Character	Character Code	Byte Sequence
5	53	53
A	65	65
m	109	109

ASCII

The diagram illustrates the mapping of characters to their ASCII codes and byte sequences. A blue oval encircles the characters '5', 'A', and 'm'. Red arrows point from each character to its corresponding code and then to its byte sequence.



Character Sets



Character Sets

- ≤ 256 characters:
 - ASCII (English)
 - ISO 8859-1 (English & Western European languages)
 - KOI8 (Cyrillic)
 - ISO-8859-6 (Arabic)
- $256 <$ characters:
 - Unicode
 - GB 2312 (Simplified Chinese)
 - Big5 (Traditional Chinese)
 - JISX 0208 (Japanese)
 - KPS 9566 (North Korean)



Character Sets

	ASCII	ISO 8859-1	ISO 8859-6	GB 2312	Unicode
A	65	65	65	65	65
ë	-	235	-	-	235
ض	-	-	214	-	1590
美	-	-	-	50112	32654
♥	-	-	-	-	9829



Unicode Standard

- History
 - ISO Universal Character Set (1989)
 - Unicode 1.0 (1991)
 - 16-bit fixed length codes.
 - Unicode 2.0 (1996)
 - Oops, we've got many more.
 - Extended to 32 bits.
 - Unicode 5.0 (2006)
 - Keep growing...



Unicode Standard

- Hexadecimal notation (**U+XXXX**).
- ISO 8859-1 is preserved as the first 256 characters.

	ISO 8859-1	Unicode
A	65	U+0041 (65)
ë	235	U+00EB (235)



Problems in Unicode

- Politics (Microsoft, Apple, Sun, ...)
- Lots of application specific characters.
 - Musical notes, circuit symbols, meteorological symbols, etc.
 - Some incomplete, others too detailed.
 - **<http://unicode.org/charts/>**
- CJK unification.



Spaces and Hyphens

Space	U+0020	Hyphen-minus (-)	U+002D
No-break Space	U+00A0	Hyphen (-)	U+2010
Em Space	U+2003	Minus (—)	U+2212
Thin Space	U+2009	Figure Dash (–)	U+2012
Zero-width Space	U+200B	Em Dash (—)	U+2014
Ideographic Space	U+3000	Quotation Dash (—)	U+2015



Different “A”s

A	U+0041	Latin
A	U+0391	Greek
A	U+0410	Cyrillic
A	U+FF21	Japanese



Problems with CJK

写	U+5199	Japanese
写	U+5199	Chinese
葛	U+845B	Japanese
葛	U+845B	Chinese
饅	U+9949	Japanese
饅	U+9949	Chinese



Inconsistency

- They “focused on semantic distinction,” but...

fi	Ligature for “f i”
a ⁴	Superscript number
┌ └	Box drawing characters

- Printing industry (e.g. Adobe) doesn't rely on Unicode.

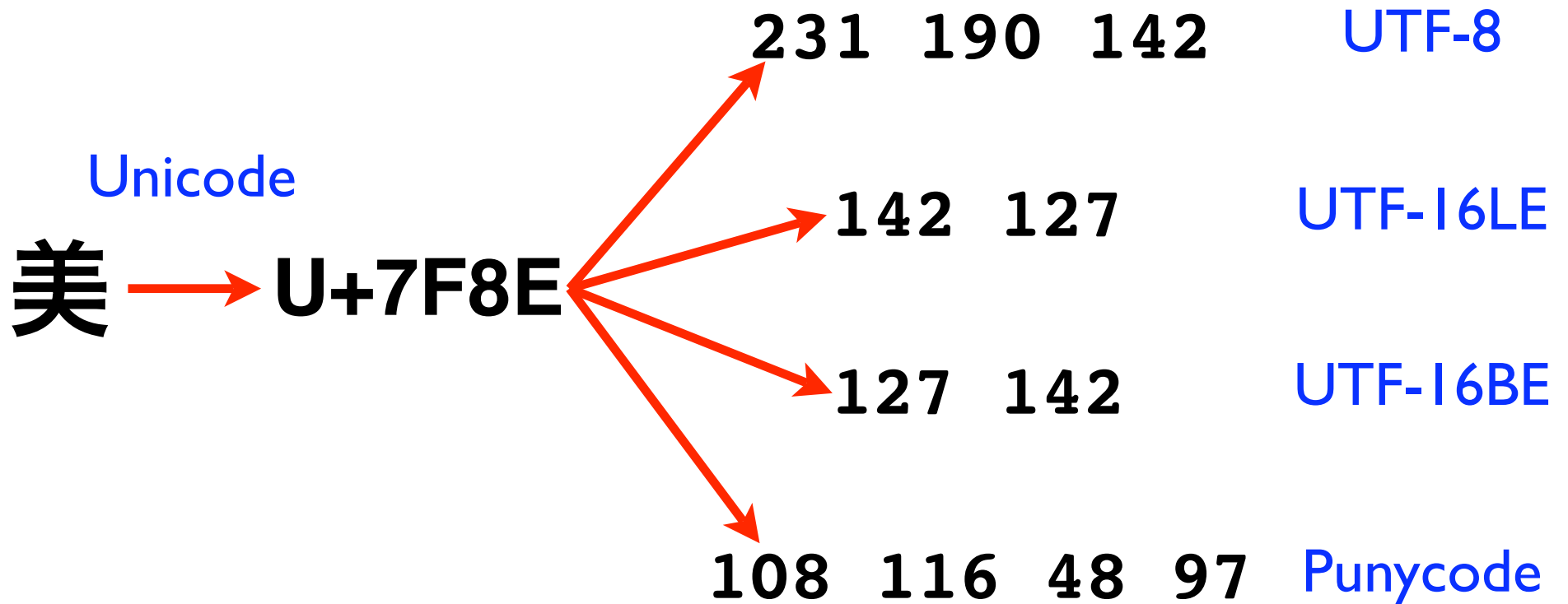


Encoding Schemes



Encodings

Same character set,
different encoding schemes.

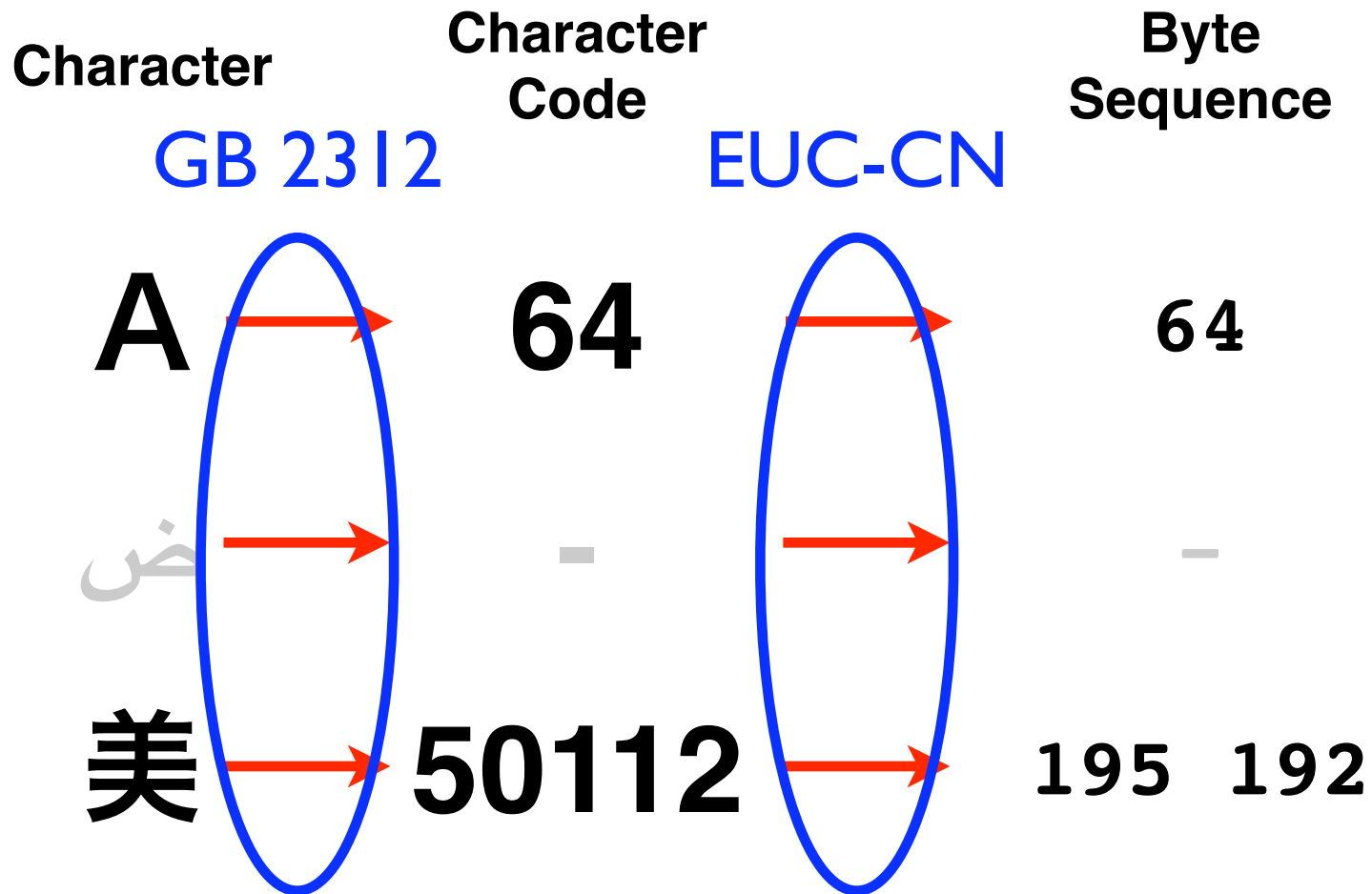


Encodings

- Most encoding schemes are associated with a certain character set.
 - **UTF-8** or **UTF-16** → Unicode
 - **EUC-CN** → GB 2312 (Simplified)
 - **Big5** → Big5 (Traditional)
 - **ISO 2022-JP** or **EUC-JP** or **Shift_JIS** → JISX 0208 (Japanese)
- Often considered representing the character set itself.



Strictly Speaking...

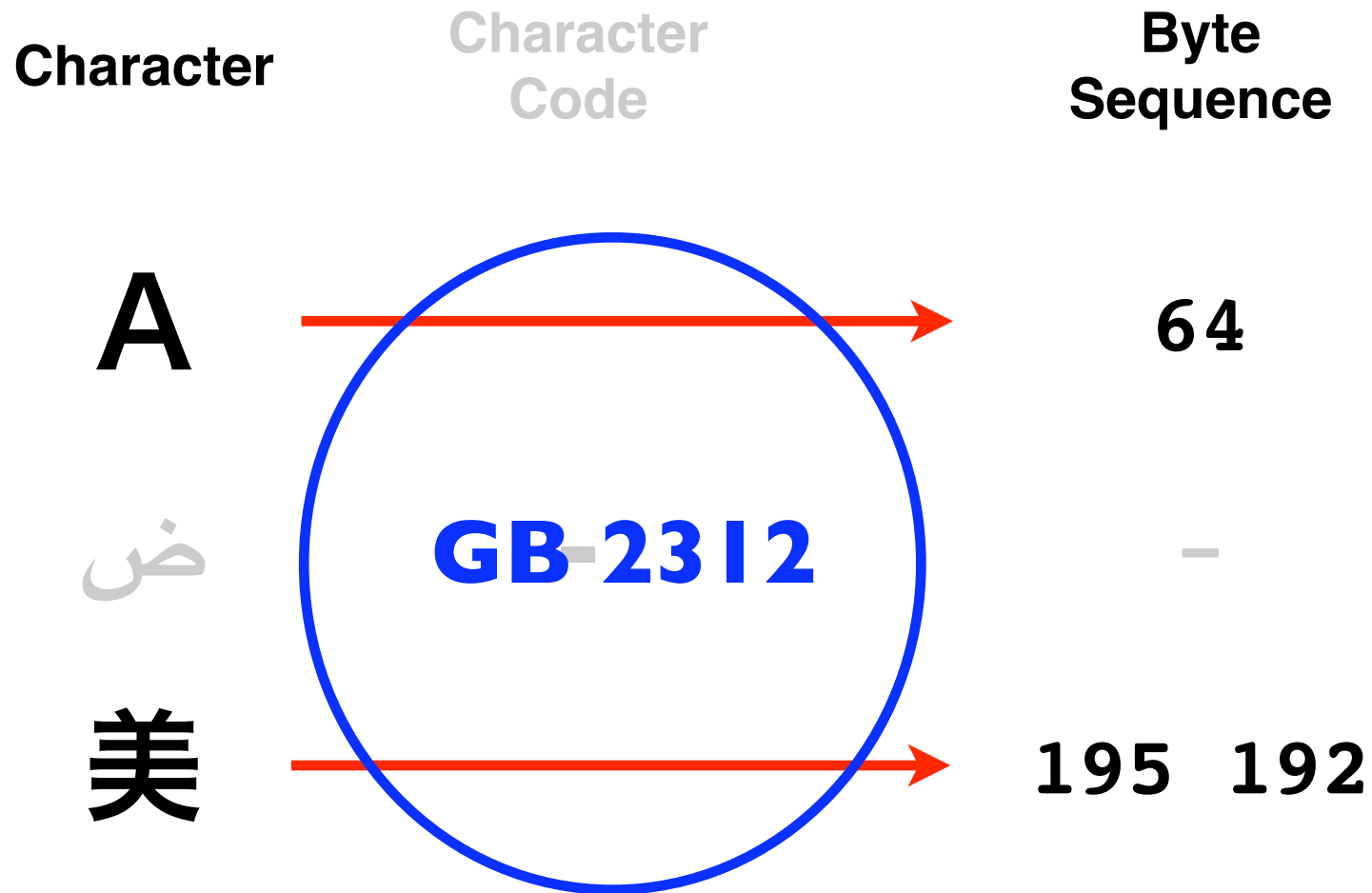


“Character Set”

“Encoding Scheme”



Often Viewed as...



Unicode Encodings

- UTF-8 (Thompson, 1992)
 - Variable length (1~6 bytes).
 - 0xxxxxxx ($U+0000 \sim U+007F$)
 - 110xxxxx 10xxxxxx ($U+0080 \sim U+087F$)
 - 1110xxxx 10xxxxxx 10xxxxxx ($U+0880 \sim U+1187F$)
 - ...
 - Easy to parse and recover.
 - MSB = 0 → an individual character.
 - MSB = 10 → a middle byte of a character.
 - MSB = 110, 1110, ... → the first byte of a character.



Unicode Encodings

- UTF-8 (Thompson, 1992)
 - ASCII characters are always represented as a single byte regardless of its context. (Good for SGML or URL)

<TEXT> □ □ □ □ □ ↵
□ □ □ □ **</TEXT>**

- As long as you only care ASCII characters and keep others untouched, your code is UTF-8 safe.



Unicode in Documents

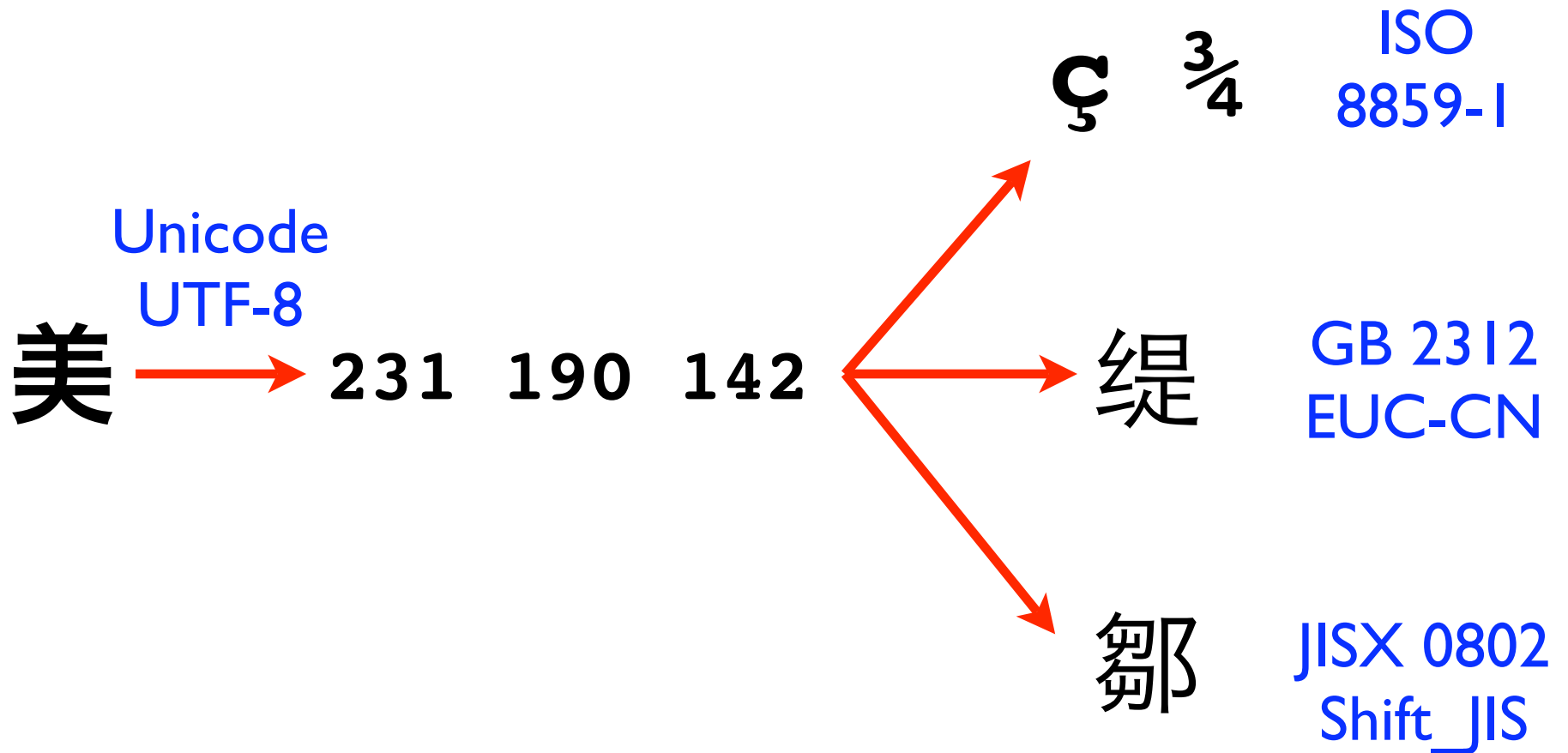


Need to Know...

- You have to know the encoding scheme (therefore its character set) of a document in advance.
- Encodings can be expressed in-document or externally.
 - HTML: `<meta>` tag, web server header.
 - XML: `<?xml?>` directive.
 - E-mail: **Content-Type**: header.



Misrecognition



HTML 4.0

- Specify the encoding

- By an external HTTP header:

```
Content-Type: text/html; charset=UTF-8
```

- By an in-document HTML tag:

```
<meta http-equiv="Content-Type"  
      content="text/html; charset=UTF-8">
```

- Use entity notation.

- Copyright © 2006 (**U+00A9** ©)
- Oct. 26, 2006 — (**U+2014** —)



XML 1.0

- Default: UTF-8
 - “Strongly recommended.”
- Can specify the encoding with a special directive.
 - `<?xml version="1.0"?>`
 - `<?xml version="1.0" encoding="iso-8859-1"?>`



Handling Unicode with Unix Tools



To View

- Firefox
 - “View” → “Character Encoding” → “Unicode”

- xterm + GNU unifont

```
$ xterm -en utf-8
```

- Less (no emulation)

```
LESSCHARSET=utf-8
```

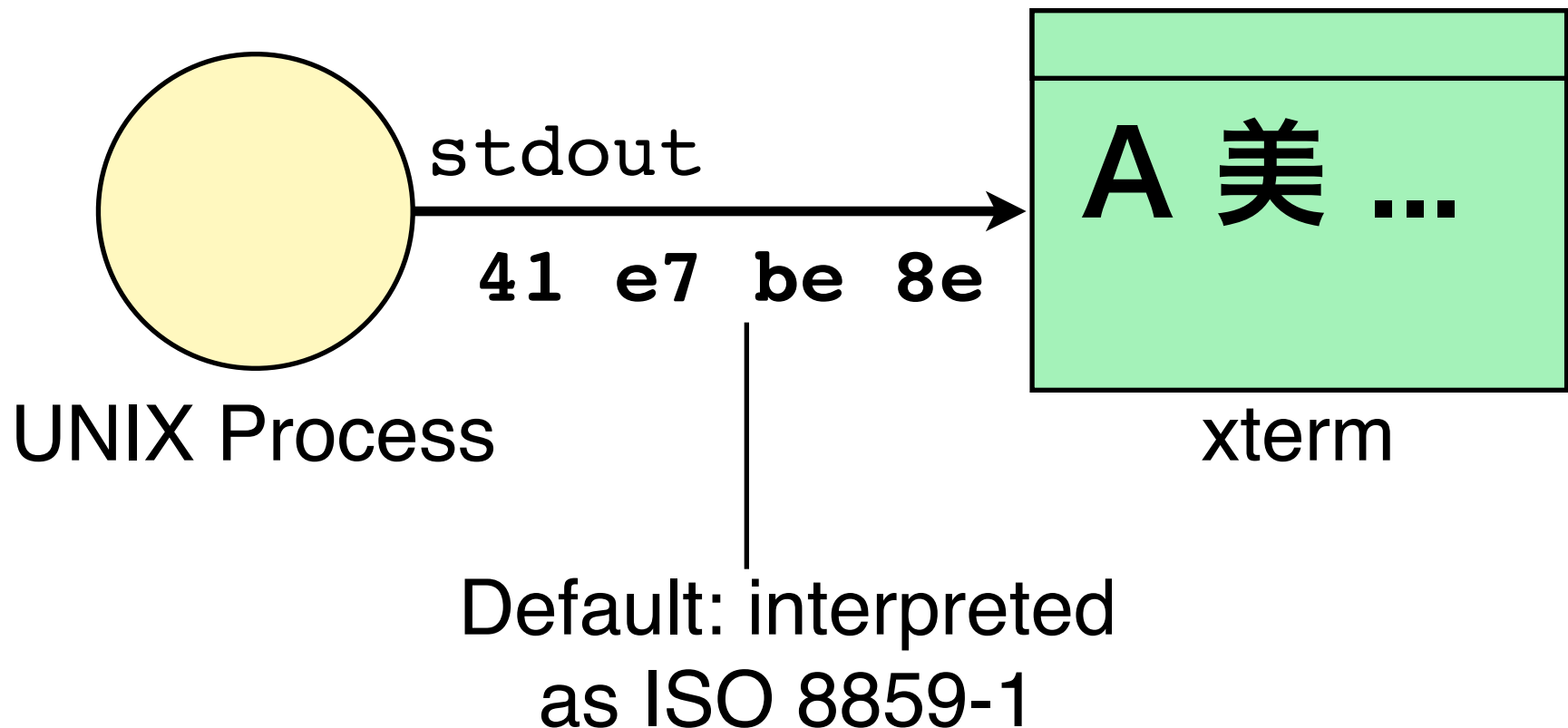
- LV (with emulation)

```
$ lv -Iu8 -Oej file.txt
```



How xterm works?

- Byte sequences are interpreted as a specific encoding.



To Edit

- Emacs + Mule-UCS
(require 'un-define)
- XEmacs
- yudit (<http://www.yudit.org>)
- gedit (buggy!)



To Convert

- `iconv`

```
$ iconv -f ISO-8859-6 -t UTF-8  
arabic > utf8
```

(Converts ISO 8859-6 into UTF-8)

- `LV`

```
$ lv -Iu8 -Oec utf8 > gb2312
```

(Converts UTF-8 into GB 2312)

- `Python`



Irreversible Nature

- Once you convert a text into Unicode, you cannot get it back to the original character set without losing information.
- Welcome to the World of the Future!



When Clueless...

- Make a guess with hex dump:

- `$ od -tx1 unknown.txt`

```
ff fe 3c 00 68 00 74 00
```

```
6d 00 6c 00 3e 00 0a 00
```

```
...
```

- UTF-16?

- BOM (U+FEFF) at the first byte.
- 00 appears many times.
- “<html>\n”



Programming for Unicode



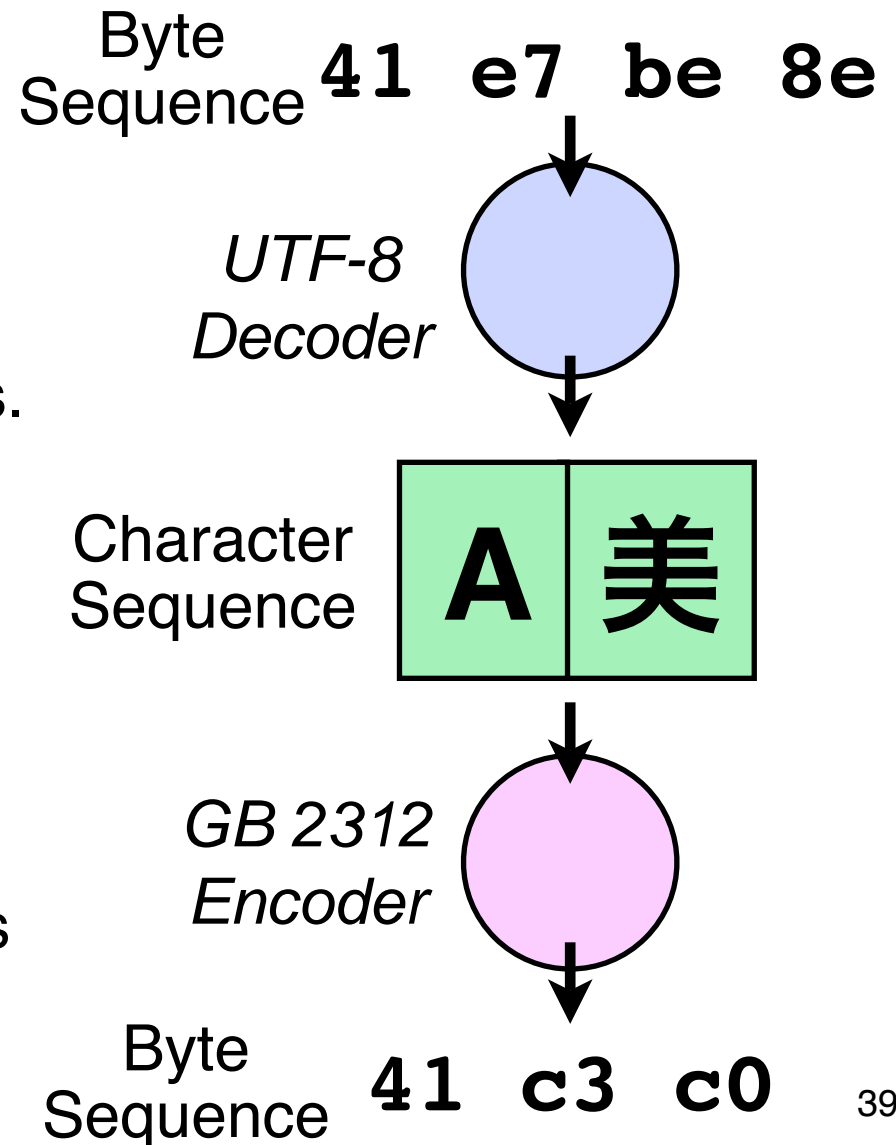
Modelling Characters

- In modern programming languages, internal/external representation of characters are separated:
 - Byte
(external representation)
 - Character
(abstract entity)
- Absorb the surface differences with encoder and decoders. ("codecs")



Modelling Characters

- Example
 - Input
 - “Decode” a UTF-8 sequence as characters.
 - Process
 - Manipulate an array of two character objects.
 - Output
 - “Encode” the characters as a GB 2312 sequence.



Java

- Two important types:
 - `String` object (Unicode string)
 - `ByteBuffer` object (byte sequence)
- Non-ASCII characters can be embedded with `\u` notation.
 - `"ABC"` (U+0041, U+0042, U+0043)
 - `"A\u7f8e"` (U+0041, U+7F8E)
美



Java

- java.io.DataInput/DataOutput
 - Only UTF-8 sequence is supported.

```
// Read UTF-8 characters.
```

```
DataInputStream in = ...;
```

```
String s = in.readUTF();
```

```
// Write UTF-8 characters.
```

```
DataOutputStream out = ...;
```

```
out.writeUTF("\u7f8e");
```



Java

- `java.nio.Charset`
 - Supports various encodings.

```
// Obtain a Charset object.
```

```
Charset cs = Charset.forName("gb2312");
```

```
// Encode the string to GB 2312.
```

```
ByteBuffer buf = cs.encode("\u7f8e");
```

```
// Decode the byte sequence as GB 2312.
```

```
String s = cs.decode(buf);
```



Python

- Two string types:
 - Byte sequence: `str. ("abc")`
 - Unicode string: `unicode. (u"abc")`

```
>>> unicode("\xe7\xbe\x8e", "utf-8")  
u'\u7f8e'      (Construct a Unicode string)
```

```
>>> u"\u7f8e".encode("utf-8")  
'\xe7\xbe\x8e' (encode in UTF-8)
```

```
>>> u"\u7f8e".encode("gb2312")  
'\xc3\xc0'    (encode in GB 2312)
```



Python

- Implicit Conversion

```
>>> print u"\u0041"
```

A *(Automatically encoded to ASCII string)*

```
>>> print u"\u7f8e" (Try to encode a non-ASCII character)
```

Traceback (most recent call last):

File "<stdin>", line 1, in ?

UnicodeEncodeError: 'ascii' codec can't
encode character u'\u7f8e' in position
0: ordinal not in range(128)



Python

- Easy to embed string constants.

```
# -*- encoding: gb2312 -*- (Magic comments)
print u"美 means beauty."
      ↓ (Automatically decoded)
      u"\u7f8e means beauty."
```

- XML parsing in 3 lines of code.

```
# parse "apf.xml" file.
fp = file("apf.xml")
root = minidom.parse(fp)
# get all <element> tags.
print root.getElementsByTagName("entity")
```



Common Lisp

- SBCL + SLIME + Mule-UCS
 - Supports UTF-8.

```
(setq slime-net-coding-system
      'utf-8-unix)
```

```
CL-USER> (code-char #x7f8e)
; => #\U7F8E
```

```
CL-USER> (coerce ' (#\U7F8E) 'string)
; => "美"
```



Conclusion

- Character sets (e.g. Unicode) and encoding schemes (e.g. UTF-8) are two different things.
- Unicode is a messy standard, but we have to get along with it.
- In modern programming languages, abstract characters and their external representation are separated.

